**Final Report: Network Traffic Anomaly Detection Using Machine Learning**

**Author:** Arish Dhingra

**Problem Statement**

The primary objective of this project is to build a system that can automatically detect anomalies in network traffic using machine learning techniques. These anomalies may include security threats such as Distributed Denial-of-Service (DDoS) attacks, scanning, brute-force attempts, and infiltration. The goal is to move from exploratory data analysis (EDA) through multiple modeling techniques to a deployable, automated solution capable of real-time detection and alerting.

**Dataset Used**

**CICIDS 2017 – Friday Working Hours Afternoon (DDoS attack)**

* This labeled dataset includes both benign and malicious traffic patterns.
* Key features include flow durations, protocol types, packet counts, byte totals, and timing-based features such as inter-arrival times.

**Data Cleaning Steps**

* Replaced invalid strings like “Infinity” or “NaN” with np.nan.
* Converted all non-numeric data to numeric format.
* Removed columns with all missing or constant values.
* Dropped features with extreme scale values (e.g., Flow Bytes/s, Fwd IAT Total) to prevent skewing the model.

**Exploratory Data Analysis (EDA)**

* Used correlation heatmaps to identify highly correlated features.
* Boxplots and distribution plots helped visualize differences between benign and malicious traffic.
* The dataset had a relatively balanced label distribution (~57% attack, ~43% benign), making it suitable for classification.

**Feature Engineering**

* Converted the Label column to binary: BENIGN = 0, all attack labels = 1.
* Applied StandardScaler to normalize all numeric input features.
* Removed redundant or constant columns to improve model stability.

**✅ Models Implemented and Results**

**Logistic Regression (Baseline)**

* LogisticRegression(max\_iter=1000)
* Accuracy: 99.85%, Precision: 1.00, Recall: 1.00, F1-Score: 1.00, ROC AUC: 0.9998

**Random Forest (Tuned with GridSearchCV)**

* Best performing model
* Accuracy: 99.99%, Precision: 1.00, Recall: 0.999, F1-Score: 0.9999, ROC AUC: 0.9999

**Isolation Forest (Unsupervised)**

* Accuracy: 34.49%, ROC AUC: 39.87%
* Struggled due to lack of labeled data and class imbalance

**Autoencoder (Deep Learning)**

* Accuracy: 40.65%, ROC AUC: 46.16%
* Based on reconstruction error thresholding

**Confusion Matrix & Evaluation Summary**

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **ROC AUC** |
| --- | --- | --- | --- | --- | --- |
| Logistic Regression | 99.85% | 99.86% | 99.88% | 99.87% | 99.85% |
| Random Forest | 99.99% | 100.00% | 99.99% | 99.99% | 99.99% |
| Isolation Forest | 34.49% | 7.45% | 1.31% | 2.23% | 39.87% |
| Autoencoder | 40.65% | 38.16% | 6.70% | 11.40% | 46.16% |

**Next Steps (Deployment Phase with Detailed Implementation Plan)**

**1. Integrate Live Network Data (Zeek or Wireshark)**

**Steps:**

* Install Zeek: sudo apt install zeek
* Enable real-time monitoring on LAN interfaces
* Parse conn.log into CSV using Python scripts
* Feed parsed files into prediction pipeline or store in S3 for processing

**2. Retrain Models Nightly and Evaluate Drift**

**Steps:**

* Schedule jobs via cron, Airflow, or AWS Batch
* Use new logs from the past 24 hours
* Save retrained models with timestamp: model\_YYYYMMDD.pkl
* Compare performance with previous model using delta in metrics or drift detection methods (e.g., KL divergence)

**3. Automate Inference in Docker**

**Steps:**

* Create a script (predict.py) that loads the model and processes incoming logs
* Dockerfile sample:

***FROM python:3.9***

***COPY . /app***

***WORKDIR /app***

***RUN pip install -r requirements.txt***

***CMD ["python", "predict.py"]***

* Build with docker build -t anomaly-detector .
* Run with docker run anomaly-detector

**4. Deploy to AWS (ECS or Lambda)**

**Steps:**

* Push Docker image to Amazon ECR
* Use ECS with Fargate for autoscaling, or Lambda for event-based triggers
* Connect API Gateway to Lambda or ECS task
* Store results in DynamoDB or S3

**5. Build Anomaly Visualization Dashboard**

**Steps:**

* Use Streamlit, Dash, or ReactJS frontend
* Visualizations:
  + Anomalies over time
  + Source/destination IP trends
  + Protocol distribution
* Host via AWS EC2, S3 + CloudFront, or Streamlit Cloud

**6. Integrate Slack Alerts**

**Steps:**

* Use AWS SNS + Lambda for alerting
* Set thresholds (e.g., more than 10 anomalies in 5 minutes)
* Lambda function sends POST to Slack webhook
* Slack alert contains IP address, confidence score, timestamp

**Contact**

* **Arish Dhingra**
* 📧 arish\_dhingra@outlook.com